

Pemrograman Berorientasi Obyek

Methods

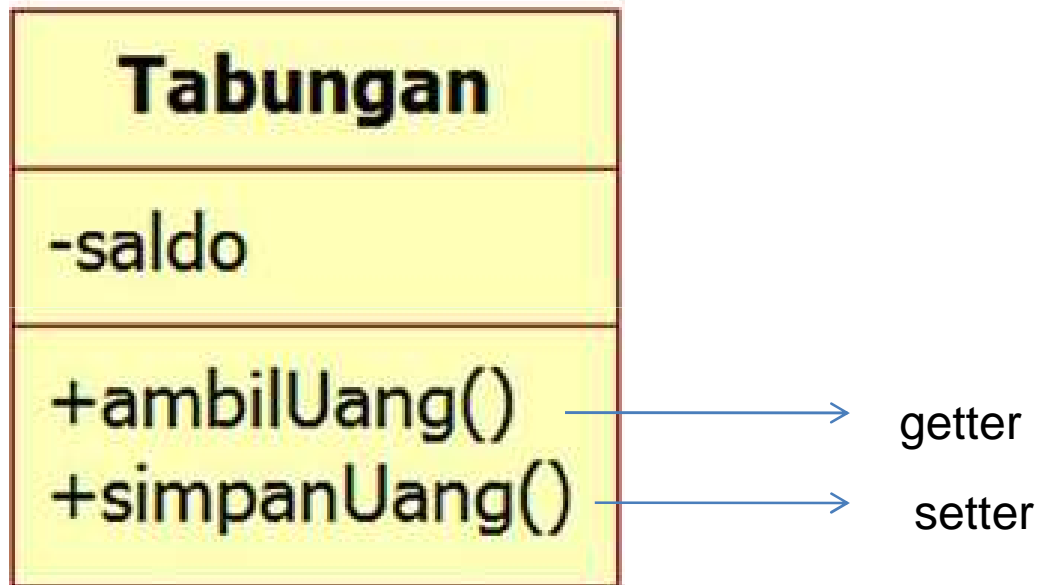
Method

- Disebut juga **behaviour**
- Memiliki hak akses **modifier** seperti pada pertemuan sebelumnya
- Terdapat 2 jenis method:
 - **Asesor**: method “pengambil nilai”, ada return value
 - getter, **get**NamaField
 - **Mutator**: method “pengubah nilai”
 - Setter, **set**NamaField

Tujuan Method

- Untuk implementasi dari enkapsulasi
- Untuk melindungi pengaksesan langsung variabel class dari luar class
- Untuk menciptakan modularitas
- Untuk menciptakan konsep API
- Getter dan Setter tidak harus ada pada semua variabel class
- *Client tidak perlu tahu detail sebuah isi variabel class dan juga isi method!*

Kasus





Contoh

```
public class Tabungan {  
    private int saldo;  
    ...  
    public int getSaldo() {  
        return saldo;  
    }  
  
    public void setSaldo(int s) {  
        if(s >= 0)  
            saldo = s;  
    }  
}
```

} **Accessor / getter**

} **Mutator / setter**

Remember *this* ?

```
public class Tabungan {  
    private double saldo;  
  
    public void ambilUang(double d) {  getter  
        double saldoBaru = saldo - d;  
        if(saldoBaru >= 0.0)  
            this.saldo = saldoBaru;  
    }  
  
    public void simpanUang(double d) {  setter  
        if(d > 0.0) {  
            this.saldo = this.saldo + d;  
        }  
    }  
}
```

Apa bedanya atribut berikut?



Implementasi Enkapsulasi

FacebookUser
-username -password -email
+setUserName() +setPassword() +setEmail()

```
FacebookUser fu = new FacebookUser()
```

```
fu.setUserName("anton");  
fu.setPassword("rahasia");  
fu.setEmail("anton@gmail.com");
```


Method invocation

- Pemanggilan method menggunakan tanda titik (.)
 - static method:
 - `namaClass.method(arguments)`
 - non-static method:
 - `namaObject.method(arguments)`

Method parameter

- Parameter yang ada pada suatu **method**
- Untuk memanggil method tersebut, kita mungkin perlu menyertakan **parameter**
- Contoh:

```
public void setName(String nama){  
    //isi kode  
}
```

Method Paramter

- Ada method yang bisa dipanggil **tanpa** membutuhkan parameter
- Contoh:

```
public String getNama(){  
    return this.nama;  
}
```

Method Name

- Dalam satu class, bisa didefinisikan dua method atau lebih dengan **nama method yang sama**
- Yang membedakan method satu dengan lainnya adalah **jumlah** dan **jenis** parameternya (signature nya)

Pengiriman parameter

- Pengiriman parameter pada Java bersifat **by value**
- Ada dua value:
 - Tipe data **primitif**
 - Yang dikirimkan adalah nilainya (by value)
 - Perubahan nilai variabel di dalam method tidak berdampak pada nilai variabel di luar method
 - Tipe data **object**
 - Yang dikirimkan adalah 'nilainya' (alamat memory object) by value
 - Perubahan object di dalam method akan mempengaruhi object tersebut

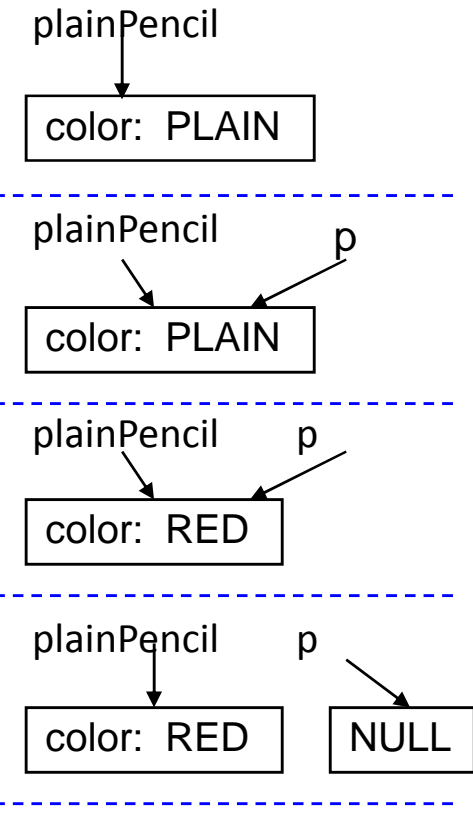
Pass By Value

```
class PassRef{
    public static void main(String[] args) {
        Pencil plainPencil = new Pencil("PLAIN");
        System.out.println("original color: " +
            plainPencil.color);

        paintRed(plainPencil);

        System.out.println("new color: " +
            plainPencil.color);
    }

    public static void paintRed(Pencil p) {
        p.color = "RED";
        p = null;
    }
}
```



- You can change which object a parameter refers to inside a method without affecting the original reference which is passed
- What is passed is the object reference, and it's passed in the manner of "PASSING BY VALUE"!

Overloading method

- `public void println(String s) { ... }`
- `public void println (int n) { ... }`
- `public void println(double d) { ... }`

- `System.out.println("Halo semua...");`
- `System.out.println(10);`
- `System.out.println(3.1415);`

Method **yang sama**

- `public int hitung(int alas, int tinggi) { ... }`
- `public int hitung(int a, int t) { ... }`
- Kedua method tersebut dianggap **sama**, anda akan **gagal** untuk kompilasi

Gagal kompilasi

```
public class Contoh {  
    public double hitung(int alas, int tinggi) {  
        double luas = 0.5 * alas * tinggi;  
        return luas;  
    }  
}
```

hitung(int,int) is already defined in Contoh
--
(Alt-Enter shows hints)

```
    public double hitung(int a, int b) {  
        double luas = a * b;  
        return luas;  
    }  
}
```

Contoh.java:17: hitung(int,int) is already defined in Contoh
 public double hitung(int a, int b) {
 ^
1 error

Method signature

- `public int hitung(int alas, int tinggi) { ... }`
- `public int hitung(int a, int t) { ... }`
- `public int hitung(int, int) -> method signature`

Method signature

- Jika ada dua method bernama sama, method **signature** keduanya harus berbeda.
- Perbedaan dilihat dari jumlah dan urutan parameter
- `public void daftar(String nama, int nim)`
- `public void daftar(int nim, String nama)`
- `public void daftar (String n, int id)`
- `public int daftar(String nama, int nim)`

Overloading konstruktor

- Satu class bisa memiliki lebih dari 1 constructor
- Constructor harus berbeda (memiliki method signature yang berbeda) satu sama lain
- Tabungan(String nama, double saldo)
- Tabungan()
- Tabungan(String nama)

Contoh

```
public class Tabungan {  
    private double saldo;  
    Tabungan(double saldo) {  
        this.saldo = saldo;  
    }  
  
    Tabungan() {  
        this.saldo = 0;  
    }  
    public void simpanUang(double d) {  
        if(d > 0.0)  
            this.saldo = this.saldo + d;  
    }  
    public double getSaldo() { return saldo; }  
}
```

Class reuse

- Jika class-class berada dalam satu direktori yang sama, class bisa langsung digunakan
- Jika ingin menggunakan class yang berada di lokasi lain, gunakan keyword **import** dan **package**
- Class-class dikelompokkan dalam **package-package**

Contoh

```
package tres;
```

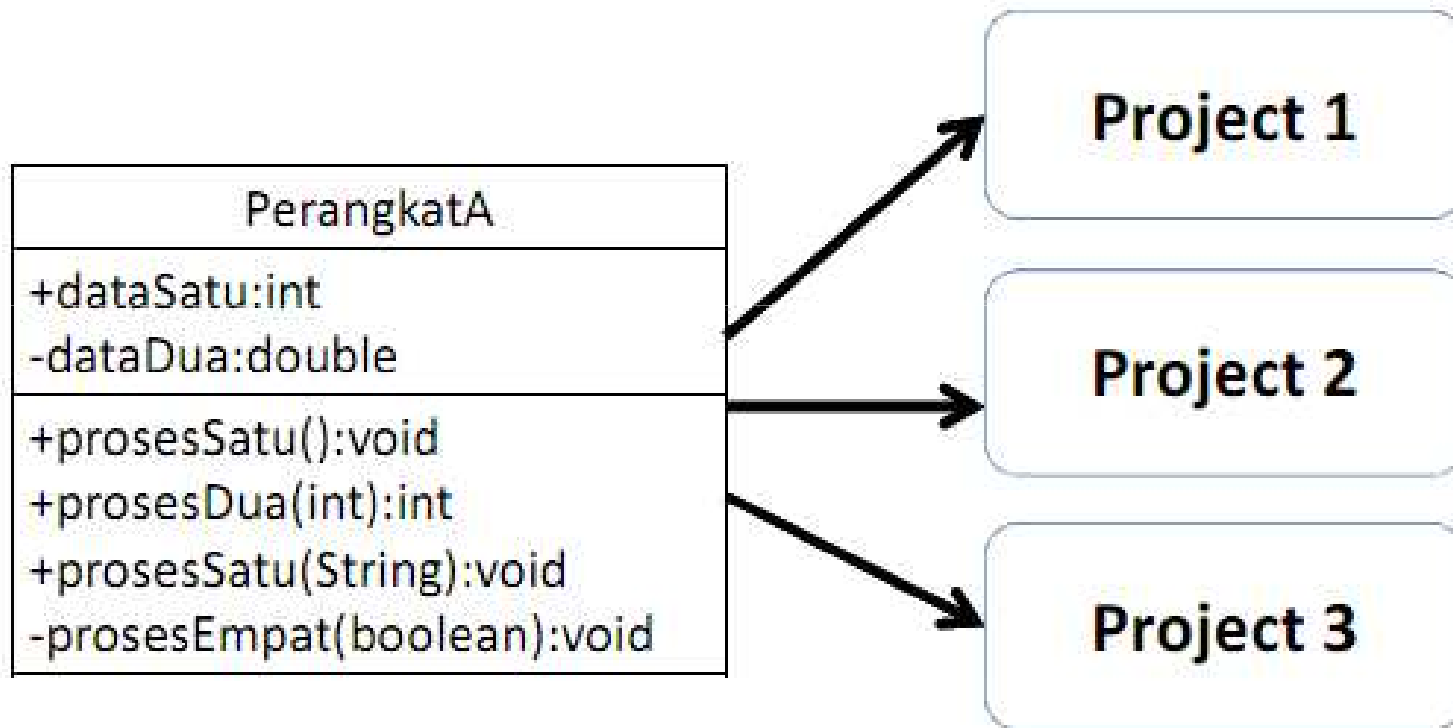
```
import dos.*;  
import uno.Beta;
```

```
Class Tester {  
    public static void main(String[] args) {  
        Delta delta = new Delta();  
        Beta beta = new Beta();  
        Sigma sigma = new Sigma();  
        ...  
    }  
}
```

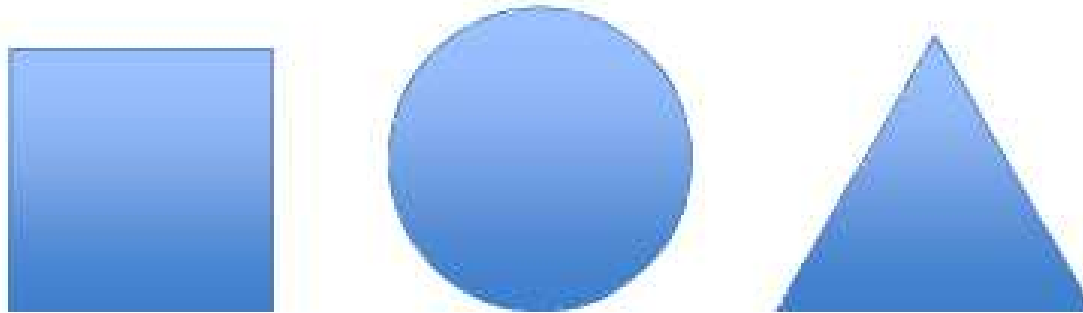
Class reuse

- Class-class yang sudah ada bisa langsung digunakan
- Suatu saat kita butuh mengubah atau menambahkan fungsi baru/atribut baru ke class yang sudah ada
- Ada beberapa pilihan:
 - Membuat class baru
 - Mengubah class lama

Contoh penggunaan



Contoh kasus



Buat aplikasi yang akan menampilkan tiga macam bentuk di layar. Masing-masing Jika diklik dengan mouse akan Berputar 180° dan memainkan file suara berformat aif spesifik untuk setiap bentuk

Contoh Kasus

```
rotate(bentuk) {  
    //rotasi 180o  
}
```

Cara Procedural !

```
playsound(bentuk) {  
    //cari suara yang akan dimainkan (.aif)  
}
```

Contoh Kasus

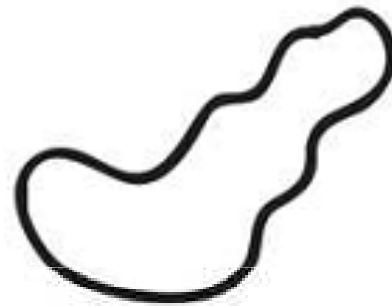
Cara OOP!

Bujursangkar
+rotate() +playSound()

Segitiga
+rotate() +playSound()

Lingkaran
+rotate() +playSound()

Perubahan requirements



- Tambah satu bentuk lagi, yaitu Amoeba !
- Jika diklik akan berotasi 180° dan memainkan suara dari file .hif

Cara prosedural

```
rotate(bentuk) {  
    //rotasi 180o  
}
```

Perubahan requirement akan membuat Anda selalu mengubah code

```
playsound(bentuk) {  
    //cari suara yang akan dimainkan  
    //jika amoeba, mainkan file .hif  
    //jika bukan, mainkan file .aif  
}
```

Cara OOP

Bujursangkar
+rotate() +playSound()

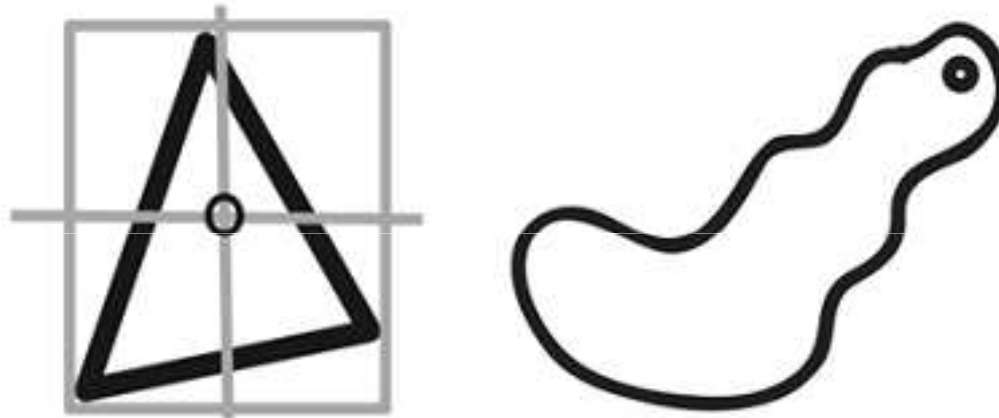
Segitiga
+rotate() +playSound()

Lingkaran
+rotate() +playSound()

Amoeba
+rotate() +playSound()

Penambahan Amoeba tidak membuat anda harus mengubah code sebelumnya yang sudah berjalan dengan baik !

Ada kesalahan!



- Ternyata bentuk Amoeba bukan diputar dengan poros titik tengah seperti bentuk lainnya

Cara prosedural

```
rotate(bentuk, x, y) {  
    //jika amoeba, rotasi x,y  
  
    //jika bukan amoeba, rotate seperti biasa  
}  
  
playsound(bentuk) {  
    //cari suara yang akan dimainkan  
    //jika amoeba, mainkan file .hif  
    //jika bukan, mainkan file .aif  
}
```

Cara OOP

Bujursangkar	Segitiga	
+rotate() +playSound()	+rotate() +playSound()	
Lingkaran		Amoeba
		-x:int -y:int
+rotate() +playSound()		+rotate() +playSound()

Untuk rotasi Amoeba, tinggal tambahkan field koordinat x dan y, lalu ubah method rotate untuk rotasi x,y

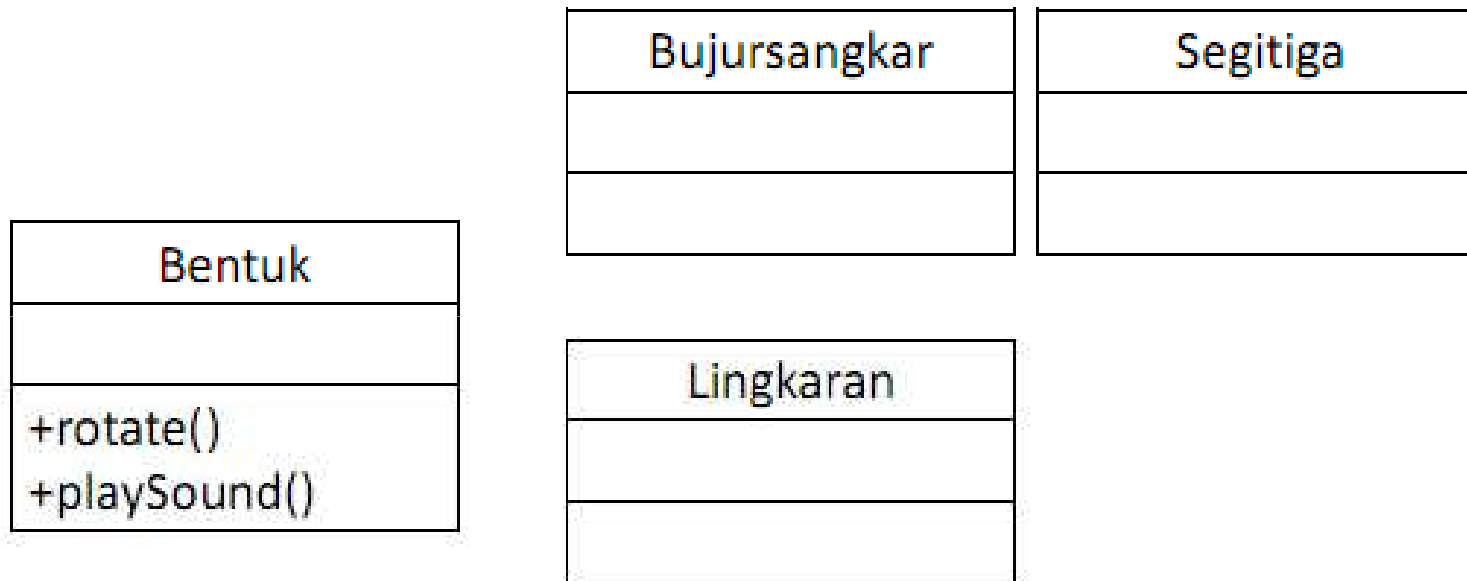
Tapi...

- Dengan cara OOP, kita punya 4 method rotate() dan 4 method playSound()
- Seluruh method tersebut harus di-kelola satu-persatu
- Pada tahap ini, design OO masih belum selesai

Perhatikan

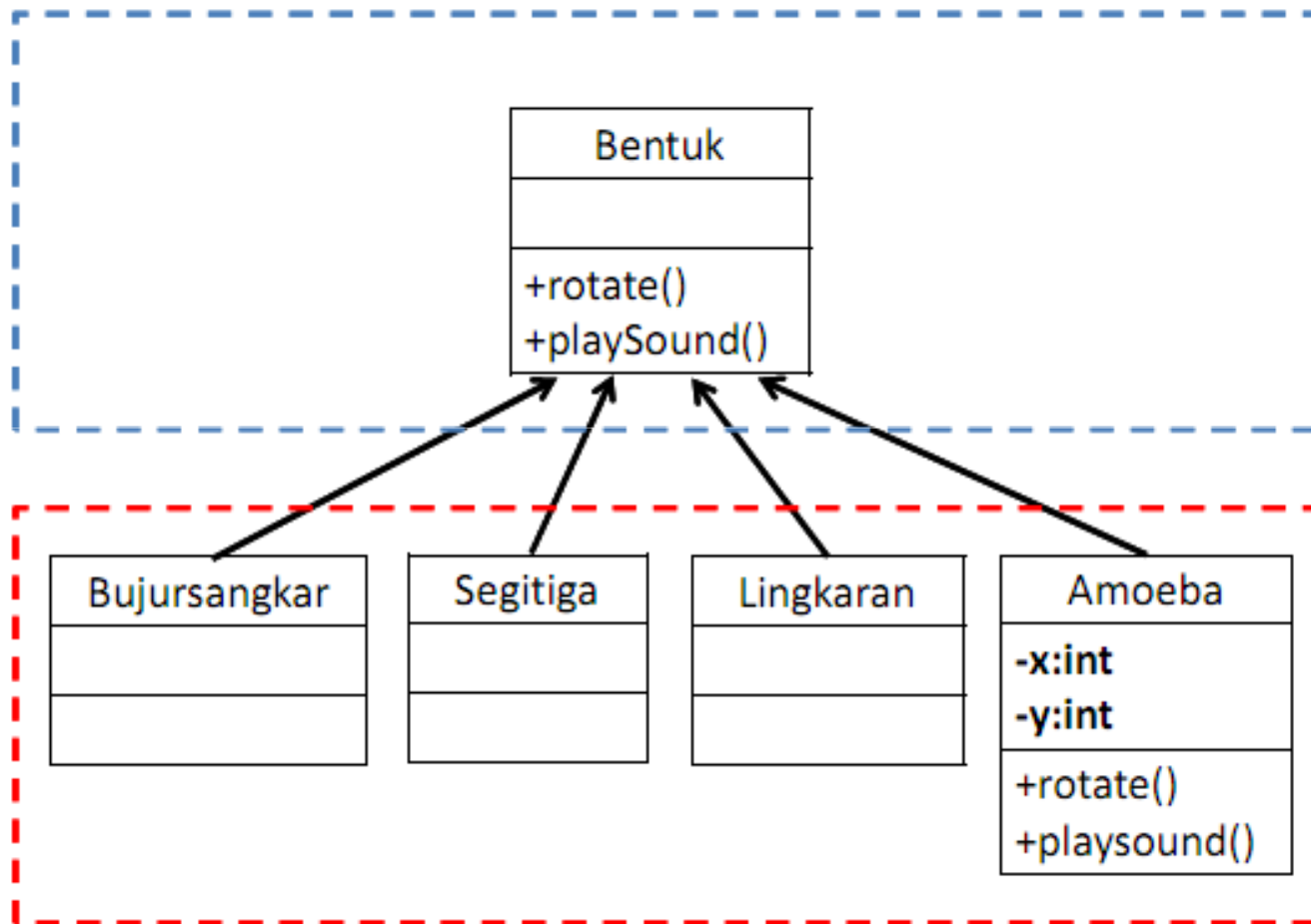
Bujursangkar	Segitiga
+rotate() +playSound()	+rotate() +playSound()
Lingkaran	Amoeba
	-x:int -y:int
+rotate() +playSound()	+rotate() +playSound()

Seluruh class memiliki method rotate() dan playSound()



Method yang sama dikelompokkan dalam class Bentuk

Akhirnya....



Akan dibahas detail di pertemuan berikutnya.....

NEXT

- Inheritance